

EVO 52702256

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**DYNAMIC DATA STRUCTURES FOR TRACKING DATA
STORED IN A FLASH MEMORY DEVICE**

Inventor(s):

Jered Aasheim

Yongqi Yang

ATTORNEY'S DOCKET NO. MS1-1070US

10087251, 022702

1 **TECHNICAL FIELD**

2 This invention relates to flash memory devices, and more particularly,
3 management of data associated with flash memory devices.
4

5 **BACKGROUND**

6 Flash memory devices have many advantages for a large number of
7 applications. These advantages include their non-volatility, speed, ease of erasure
8 and reprogramming, small physical size and related factors. There are no
9 mechanical moving parts and as a result such systems are not subject to failures of
10 the type most often encountered with hard disk storage systems. As a result many
11 portable computer devices, such as laptops, portable digital assistants, portable
12 communication devices, and many other related devices are using flash memory as
13 the primary medium for storage of information.

14 Flash memory devices are generally operated by first setting all bits in a
15 block to a common state, and then reprogramming them to a desired new state.
16 Blocks of data need to be shuffled during the reprogramming process, which can
17 slow the completion of the operation. Besides being time consuming,
18 reprogramming a block of data can subject the entire block to accidental loss, in
19 the event there is a power failure during the reprogramming process. Normally, as
20 the block is shuffled, it is temporarily stored in a volatile memory device, such as
21 Random Access Memory (RAM). The entire block of data (not just newly entered
22 data) is susceptible to permanent loss if the reprogramming process has not
23 completed prior to the power failure. In these circumstances, an entire block of
24 data may need to be reentered by a user anew.
25

10087251.022702

1
2 **SUMMARY**

3 A dynamic data structure for tracking data stored in a Flash memory device
4 is described. In a described implementation, a process maintains one or more
5 mapping data structures containing mappings of logical flash memory addresses to
6 physical flash memory addresses. Each mapping data structure has a
7 predetermined capacity of mappings. The process also maintains a master data
8 structure containing a pointer to each of the one or more mapping data structures.
9 Additional mapping data structures are allocated as needed to provide capacity for
10 additional mappings. Each time a mapping data structure is allocated or de-
11 allocated the pointers in the master data structure are changed accordingly.

12 The described implantations, therefore, introduce the broad concept of
13 using a dynamic amount of memory to store logical-to-physical sector address
14 mappings. Thus, the amount of memory needed to track data stored in the flash
15 memory medium can be minimized.

16
17 **BRIEF DESCRIPTION OF THE DRAWINGS**

18 The detailed description is described with reference to the accompanying
19 figures. In the figures, the left-most digit(s) of a reference number identifies the
20 figure in which the reference number first appears.

21 FIG. 1 illustrates a logical representation of a NAND flash memory
22 medium.

23 FIG. 2 illustrates a logical representation of a NOR flash memory medium.

24 FIG. 3 illustrates pertinent components of a computer device, which uses
25 one or more flash memory devices to store information.

1 FIG. 4 illustrates a block diagram of flash abstraction logic.

2 FIG. 5 illustrates an exemplary block diagram of a flash medium logic.

3 FIG. 6A shows a data structure used to store a corresponding relationship
4 between logical sector addresses and physical sector addresses.

5 FIG. 6B shows a data structure which is the same as the data structure in
6 FIG. 6B, except its contents have been updated.

7 FIG. 7 illustrates a process used to track data on the flash memory medium
8 when the file system issues write requests to the flash driver.

9 FIG. 8 illustrates a process for safeguarding mapping of logical-to-physical
10 sector address information stored in volatile data structures, such as the data
11 structures shown in FIGS. 6A and 6B.

12 FIG. 9 illustrates a location within the flash memory medium in which the
13 logical sector address can be stored for safeguarding in the event of a power
14 failure.

15 FIG. 10 illustrates a dynamic look-up data structure to track data stored in
16 the flash memory medium.

17 FIG. 11 illustrates a process for dynamically allocating look-up data
18 structures for tracking data on the flash memory medium.

19 FIG. 12 is a diagram of the flash memory medium viewed and/or treated as
20 a continuous circle by the flash driver.

21 FIG. 13 depicts another illustration of the media viewed as a continuous
22 circle.

23 FIG. 14 illustrates a process used by the sector manager to determine the
24 next available free sector location for the flash driver to store data on the medium.

25 FIG. 15 illustrates another view of media treated as a continuous circle.

1 FIG. 16 is a flow chart illustrating a process used by the compactor to
2 recycle sectors.

3 FIG. 17 shows one exemplary result from the process illustrated in FIG. 16.

4 FIG. 18 illustrates a logical representation of a NOR flash memory medium
5 divided in way to better support the processes and techniques implemented by the
6 flash driver.

7 8 **DETAILED DESCRIPTION**

9 The following discussion is directed to flash drivers. The subject matter is
10 described with specificity to meet statutory requirements. However, the
11 description itself is not intended to limit the scope of this patent. Rather, the
12 inventors have contemplated that the claimed subject matter might also be
13 embodied in other ways, to include different elements or combinations of elements
14 similar to the ones described in this document, in conjunction with other present or
15 future technologies.

16 ***Overview***

17 This discussion assumes that the reader is familiar with basic operating
18 principles of flash memory media. Nevertheless, a general introduction to two
19 common types of nonvolatile random access memory, NAND and NOR Flash
20 memory media, is provided to better understand the exemplary implementations
21 described herein. These two example flash memory media were selected for their
22 current popularity, but their description is not intended to limit the described
23 implementations to these types of flash media. Other electrically erasable and
24 programmable read-only memories (EEPROMs) would work too. In most
25

examples used throughout this Detailed Description numbers shown in data structures are in decimal format for illustrative purposes.

Universal Flash Medium Operating Characteristics

FIG. 1 and FIG. 2 illustrate logical representations of example NAND and NOR flash memory media 100, 200, respectively. Both media have universal operating characteristics that are common to each, respectively, regardless of the manufacturer. For example referring to FIG. 1, a NAND flash memory medium is generally split into contiguous blocks (0, 1, through N). Each block 0, 1, 2, etc. is further subdivided into K sectors 102; standard commercial NAND flash media commonly contain 8, 16, or 32 sectors per block. The amount of blocks and sectors can vary, however, depending on the manufacturer. Some manufacturers refer to “sectors” as “pages.” Both terms as used herein are equivalent and interchangeable.

Each sector 102 is further divided into two distinct sections, a data area 103 used to store information and a spare area 104 which is used to store extra information such as error correction code (ECC). The data area 103 size is commonly implemented as 512 bytes, but again could be more or less depending on the manufacturer. At 512 bytes, the flash memory medium allows most file systems to treat the medium as a nonvolatile memory device, such as a fixed disk (hard drive). As used herein RAM refers generally to the random access memory family of memory devices such as DRAM, SRAM, VRAM, VDO, and so forth. Commonly, the size of the area spare 104 is implemented as 16 bytes of extra storage for NAND flash media devices. Again, other sizes, greater or smaller can

1 be selected. In most instances, the spare area 104 is used for error correcting
2 codes, and status information.

3 A NOR memory medium 200 is different than NAND memory medium in
4 that blocks are not subdivided into physical sectors. Similar to RAM, each byte
5 stored within a block of NOR memory medium is individually addressable.
6 Practically, however, blocks on NOR memory medium can logically be
7 subdivided into physical sectors with the accompanying spare area.

8 Aside from the overall layout and operational comparisons, some universal
9 electrical characteristics (also referred to herein as “memory requirements” or
10 “rules”) of flash devices can be summarized as follows:

- 11 1. Write operations to a sector can change an individual bit from a
12 logical ‘1’ to a logical ‘0’, but not from a logical ‘0’ to logical ‘1’ (except for case
13 No. 2 below);
- 14 2. Erasing a block sets all of the bits in the block to a logical ‘1’;
- 15 3. It is not generally possible to erase individual sectors/bytes/bits in a
16 block without erasing all sectors/bytes within the same block;
- 17 4. Blocks have a limited erase lifetime of between approximately
18 100,000 to 1,000, 000 cycles;
- 19 5. NAND flash memory devices use ECC to safeguard against data
20 corruption due to leakage currents; and
- 21 6. Read operations do not count against the write/erase lifetime.

22 *Flash Driver Architecture*

24 FIG. 3 illustrates pertinent components of a computer device 300, which
25 uses one or more flash memory devices to store information. Generally, various

1 different general purpose or special purpose computing system configurations can
2 be used for computer device 300, including but not limited to personal computers,
3 server computers, hand-held or laptop devices, portable communication devices,
4 multiprocessor systems, microprocessor systems, microprocessor-based systems,
5 programmable consumer electronics, gaming systems, multimedia systems, the
6 combination of any of the above example devices and/or systems, and the like.

7 Computer device 300 generally includes a processor 302, memory 304, and
8 a flash memory media 100/200. The computer device 300 can include more than
9 one of any of the aforementioned elements. Other elements such as power
10 supplies, keyboards, touch pads, I/O interfaces, displays, LEDs, audio generators,
11 vibrating devices, and so forth are not shown, but could easily be a part of the
12 exemplary computer device 300.

13 Memory 304 generally includes both volatile memory (e.g., RAM) and
14 non-volatile memory (e.g., ROM, PCMCIA cards, etc.). In most implementations
15 described below, memory 304 is used as part of computer device's 302 cache,
16 permitting application data to be accessed quickly without having to permanently
17 store data on a non-volatile memory such as flash medium 100/200.

18 An operating system 309 is resident in the memory 304 and executes on the
19 processor 302. An example operating system implementation includes the
20 Windows®CE operating system from Microsoft Corporation, but other operation
21 systems can be selected from one of many operating systems, such as DOS,
22 UNIX, etc. For purposes of illustration, programs and other executable program
23 components such as the operating system are illustrated herein as discrete blocks,
24 although it is recognized that such programs and components reside at various
25

20220725 10:37:25

1 times in different storage components of the computer, and are executed by the
2 processor(s) of the computer device 300.

3 One or more application programs 307 are loaded into memory 304 and run
4 on the operating system 309. Examples of applications include, but are not limited
5 to, email programs, word processing programs, spreadsheets programs, Internet
6 browser programs, as so forth.

7 Also loaded into memory 304 is a file system 305 that also runs on the
8 operating system 309. The file system 305 is generally responsible for managing
9 the storage and retrieval of data to memory devices, such as magnetic hard drives,
10 and this exemplary implementation flash memory media 100/200. Most file
11 systems 305 access and store information at a logical level in accordance with the
12 conventions of the operating system the file system 305 is running. It is possible
13 for the file system 305 to be part of the operating system 309 or embedded as code
14 as a separate logical module.

15 Flash driver 306 is implemented to function as a direct interface between
16 the file system 305 and flash medium 100/200. Flash driver 306 enables computer
17 device 300 through the file system 305 to control flash medium 100/200 and
18 ultimately send/retrieve data. As shall be described in more detail, however, flash
19 driver 306 is responsible for more than read/write operations. Flash driver 306 is
20 implemented to maintain data integrity, perform wear-leveling of the flash
21 medium, minimize data loss during a power interruption to computer device 300
22 and permit OEMs of computer devices 300 to support their respective flash
23 memory devices regardless of the manufacturer. The flash driver 306 is file
24 system agnostic. That means that the flash driver 306 supports many different
25 types of files systems, such as File Allocation Data structure File System (FAT16),

1 (FAT32), and other file systems. Additionally, flash driver 306 is flash memory
2 medium agnostic, which likewise means driver 306 supports flash memory
3 devices regardless of the manufacturer of the flash memory device. That is, the
4 flash driver 306 has the ability to read/write/erase data on a flash medium and can
5 support most, if not all, flash devices.

6 In the exemplary implementation, flash driver 306 resides as a component
7 within operating system 309, that when executed serves as a logical interface
8 module between the file system 305 and flash medium 100/200. The flash driver
9 306 is illustrated as a separate box 306 for purposes of demonstrating that the flash
10 driver when implemented serves as an interface. Nevertheless, flash driver 306
11 can reside in other applications, part of the file system 305 or independently as
12 separate code on a computer-readable medium that executes in conjunction with a
13 hardware/firmware device.

14 In one implementation, flash driver 306 includes: a flash abstraction logic
15 308 and a programmable flash medium logic 310. Flash abstraction logic 308 and
16 programmable medium logic 310 are coded instructions that support various
17 features performed by the flash driver 306. Although the exemplary
18 implementation is shown to include these two elements, various features from
19 each of the flash abstraction logic 308 and flash medium logic 310 may be
20 selected to carry out some of the more specific implementations described below.
21 So while the described implementation shows two distinct layers of logic 308/310,
22 many of the techniques described below can be implemented without necessarily
23 requiring all or a portion of the features from either layer of logic. Furthermore,
24 the techniques may be implemented without having the exact division of
25 responsibilities as described below.

1 In one implementation, the Flash abstraction logic 308 manages those
2 operating characteristics that are universally common to flash memory media.
3 These universal memory requirements include wear-leveling, maintaining data
4 integrity, and handling recovery of data after a power failure. Additionally, the
5 flash abstraction logic 308 is responsible for mapping information stored at a
6 physical sector domain on the flash memory medium 100/200 to a logical sector
7 domain associated with the file system 305. That is, the flash abstraction logic
8 308 tracks data going from a logical-to-physical sector addresses and/or from a
9 physical-to-logical sector addresses. Driver 306 uses logical-to-physical sector
10 addresses for both read/write operations. Driver 306 goes from physical-to-logical
11 sector addresses when creating a look-up table (to be described below) during
12 driver initialization. Some of the more specific commands issued by the file
13 system that are dependent upon a certain type of flash memory media are sent
14 directly to the flash medium logic 310 for execution and translation. Thus, the
15 flash abstraction logic 308 serves as a manager to those universal operations,
16 which are common to flash memory media regardless of the manufacturer for the
17 media, such as wear-leveling, maintaining data integrity, handling data recovery
18 after a power failure and so forth.

19 FIG. 4 illustrates an exemplary block diagram of the flash abstraction logic
20 308. Flash abstraction logic 308 includes a sector manager 402, a logical-to-
21 physical sector mapping module 404, and a compactor 406. Briefly, the sector
22 manager 402 provides a pointer to a sector available, i.e., "free" to receive new
23 data. The logical-to-physical sector mapping module 404 manages data as it goes
24 from a file system domain of logical sector addressing to a flash medium domain
25 of physical sector addressing. The compactor 406 provides a mechanism for

1 clearing blocks of data (also commonly referred to in the industry as “erasing”) to
2 ensure that enough free sectors are available for writing data. Additionally, the
3 compactor 406 helps the driver 306 system perform uniform and even wear
4 leveling. All these elements shall be described in more detail below.

5 Referring back to FIG. 3, the flash medium logic 310 is used to translate
6 logical commands, received from either the flash abstraction logic 308 or file
7 system 305, to physical sector commands for issuance to the flash memory
8 medium 100/200. For instance, the flash medium logic 310 reads, writes, and
9 erases data to and/or from the flash memory medium. The flash medium logic 310
10 is also responsible for performing ECC (if necessary). In one implementation, the
11 flash medium logic 310 is programmable to permit users to match particular flash
12 medium requirements of a specific manufacturer. Thus, the flash medium logic
13 310 is configured to handle specific nuances, ECC, and specific commands
14 associated with controlling physical aspects of flash medium 100/200.

15 FIG. 5 illustrates an exemplary block diagram of the flash medium logic
16 310. As shown, the flash medium logic 310 includes a programmable entry point
17 module 502, I/O module 504 and an ECC module 506. The programmable entry
18 point module 502 defines a set of programming interfaces to communicate
19 between flash abstraction logic 308 and flash medium 100/200. In other words, the
20 programmable entry points permit manufacturers of computer devices 300 to
21 program the flash media logic 310 to interface with the actual flash memory
22 medium 100/200 used in the computer device 300. The I/O module 504 contains
23 specific code necessary for read/write/erase commands that are sent to the Flash
24 memory medium 100/200. The user can program the ECC module 506 to function
25 in accordance with any particular ECC algorithm selected by the user.

Tracking Data

File system 305 uses logical sector addressing to read and store information on flash memory medium 100/200. Logical sector addresses are address locations that the file system reads and writes data to. They are “logical” because they are relative to the file system. In actuality, data may be stored in completely different physical locations on the flash memory medium 100/200. These physical locations are referred to as physical sector addresses.

The flash driver 306 is responsible for linking all logical sector address requests (i.e., read & write) to physical sector address requests. The process of linking logical-to-physical sector addresses is also referred to herein as mapping. Going from logical to physical sector addresses permits flash driver 306 to have maximum flexibility when deciding where to store data on the flash memory medium 100/200. The logical-to-physical sector mapping module 404 permits data to be flexibly assigned to any physical location on the flash memory medium, which provides efficiency for other tasks, such as wear-leveling and recovering from a power failure. It also permits the file system 305 to store data in the fashion it is designed to do so, without needing intelligence to know that the data is actually being stored on a flash medium in a different fashion.

FIG. 6A shows an exemplary implementation of a data structure (i.e., a table) 600A generated by the flash driver 306. The data structure 600A is stored in a volatile portion of memory 304, e.g. RAM. The data structure 600A includes physical sector addresses 602 that have a corresponding logical sector address 604. An exemplary description of how table 600A is generated is described with reference to FIG. 7.

FIG. 7 illustrates a process 700 used to track data on the flash memory medium 100/200 when the file system 305 issues write requests to the flash driver 306. Process 700 includes steps 702-718. Referring to FIGS. 6A and 7, in step 702, flash abstraction logic 308 receives a request to write data to a specified logical sector address 604.

In step 704, the sector manager 402 ascertains a free physical sector address location on the flash medium 100/200 that can accept data associated with the write request (how the sector manager 402 chooses physical sector addresses will be explained in more detail below). A free physical sector is any sector that can accept data without the need to be erased first. Once the sector manager 402 receives the physical sector address associated with a free physical sector location, the logical-to-physical sector mapping module 404 assigns the physical sector address to the logical sector address 604 specified by write request forming a corresponding relationship. For example, a physical sector address of 0 through N can be assigned to any arbitrary logical sector address 0 through N.

Next, in step 706, the logical-to-physical sector mapping module 404 stores the corresponding relationship of the physical sector address to the logical sector address in a data structure, such as the exemplary table 600A in memory 305. As shown in the exemplary data structure 600A, three logical sector addresses 604 are assigned to corresponding physical sector addresses 602.

Next, in step 708 data associated with the logical sector address write request is stored on the flash medium 100/200 at the physical sector address location assigned in step 704. For example, data would be stored in physical sector address location of zero on the medium 100/200, which corresponds to the logical sector address of 11.

1 Now, in step 710, suppose for example purposes the file system 305 issues
2 another write request, but in this case, to modify data associated with a logical
3 sector address previously issued in step 702. Then, flash driver 306 performs
4 steps 712 through 714, which are identical to steps 704 through 708, respectively,
5 which are described above.

6 In step 718, however, after the updated data associated with step 710 is
7 successfully stored on the flash medium 100/200, the logical-to-physical sector
8 mapping module 404 marks the old physical sector address assigned in step 704 as
9 "dirty." Old data is marked dirty after new data is written to the medium 100/200,
10 so in the event there is a power failure in the middle of the write operation, the
11 logical-to-physical sector mapping module 404 will not lose old data. It is
12 possible to lose new or updated data from steps 702 or 710, but since there is no
13 need to perform an erase operation only one item of new or modified data is lost in
14 the event of a power failure.

15 FIG. 6B shows a data structure 600B which is the same as data structure
16 600A, except its contents have been updated. In this example the file system 305
17 has updated data associated with logical sector address 11. Accordingly, the flash
18 driver 306 reassigns logical sector address 11 to physical sector address 3 and
19 stores the reassigned corresponding relationship between the these two addresses
20 in data structure 600B. As illustrated in data structure 600B, the contents of
21 logical sector 11 are actually written to physical sector address 3 and the contents
22 of sector 0 are marked "dirty" after the data contents are successfully written into
23 physical sector address 3 as was described with reference to steps 710-718.

24 This process of reassigning logical-to-physical sector address when
25 previously stored data is updated by the file system 305, permits write operations

1 to take place without having to wait to move an entire block of data and perform
2 an erase operation. So, process 700 permits the data structure to be quickly
3 updated and then the physical write operation can occur on the actual physical
4 medium 100/200. Flash abstraction logic 308 uses the data structures, such as
5 600A/600B, to correctly maintain logical-to-physical mapping relationships.

6 When there is a read request issued by the files system 305, the flash
7 abstraction logic 308, through the logical-to-physical mapping module 404,
8 searches the data structure 600A/600B to obtain the physical sector address which
9 has a corresponding relationship with the logical sector address associated with
10 read request. The flash medium logic 310 then uses that physical sector address as
11 a basis to send data associated with the read request back to the file system 305.
12 The file system 305 does not need intelligence to know that its requests to logical
13 sector addresses are actually mapped to physical sector addresses.

14 ***Power-Interruption Protection***

15 Write operations are performed at the sector-level as opposed to the block-
16 level, which minimizes the potential for data loss during a power-failure situation.
17 A sector worth of data is the finest level of granularity that is used with respect to
18 most file systems 305. Therefore, if the flash driver 306 is implemented to operate
19 at a per sector basis, the potential for data loss during a power failure is reduced.

20 As mentioned above, data structures 600A, 600B are stored in memory 304,
21 which in one exemplary implementation is typically a volatile memory device
22 subject to complete erasure in the event of a power failure. To safeguard data
23 integrity on the flash medium 100/200, logical-to-physical mapping information
24
25

1 stored in the data structures 600A/ 600B is backed-up on the flash memory
2 medium.

3 In one exemplary implementation, to reduce the cost associated with
4 storing the entire data structure on the flash memory medium 100/200, the logical
5 sector address is stored in the spare 104 area of the medium with each physical
6 sector in which the logical sector address has a corresponding relationship.

7 FIG. 8 illustrates a process 800 for safeguarding mapping of logical-to-
8 physical sector address information stored in volatile data structures, such as
9 exemplary data structures 600A and 600B. Process 800 includes steps 802-814.
10 The order in which the process is described is not intended to be construed as a
11 limitation. Furthermore, the process can be implemented in any suitable hardware,
12 software, firmware, or combination thereof. In step 802, the logical sector address
13 associated with the actual data is stored in the physical sector of the flash memory
14 medium 100/200 at the physical sector address assigned to the logical sector
15 address. In the case of a NAND flash memory medium 100, the logical sector
16 address is stored in the spare area 104 of the medium. Using this scheme, the
17 logical-to-physical sector mapping information is stored in a reverse lookup
18 format. Thus, after a power failure situation, it is necessary to scan the spare area
19 for each physical sector on the media, determine the corresponding logical sector
20 address, and then update the in-memory lookup table accordingly. FIG. 9
21 illustrates a location with in media 100/200 in which the logical sector address can
22 be stored. As previously mentioned, blocks of NOR flash memory can be
23 logically subdivided into physical sectors each with a spare area (similar to
24 NAND). Using this technique, the logical sector address is stored in the spare area
25

1 for each the physical sector similar to the process used with NAND flash memory
2 (shown in FIG. 15 as space 1504 to be described with reference to FIG. 15).

3 In the event there is a power interruption and the data structures 600A,
4 600B are lost, as indicated by the YES branch of decisional step 804 of FIG. 8,
5 then flash abstraction logic 308 uses the flash medium logic 310 to scan the flash
6 memory medium to locate the logical sector address stored with data in each
7 physical address (see FIG. 9), as indicated in step 806. In step 808, the physical
8 sector address in which data is contained is reassigned to the logical sector address
9 located with the data on the medium. As the physical and logical sector address
10 are reestablished they are stored back in the data structures 600A, 600B and the
11 flash medium logic 310 goes to the next sector containing data as indicated in step
12 812. Steps 806-812 repeat until all sectors containing data have been are scanned
13 and the data structure is reestablished. Normally, this occurs at initialization of the
14 computer device 300.

15 Accordingly, when a power failure occurs, process 800 enables the flash
16 abstraction logic 308 to scan the medium 100/200 and rebuild the logical-to-
17 physical mapping in a data structure such as the exemplary data structure 600.
18 Process 800 ensures that mapping information is not lost during a power failure
19 and that integrity of the data is retained.

20 *Dynamic Look-up Data structure for Tracking Data*

21 FIG. 10 illustrates a dynamic look-up data structure 1000 to track data
22 stored in the flash memory medium 100/200. Data structure 1000 includes a
23 master data structure 1002 and one or more secondary data structures 1004, 1006.
24 The data structures are generated and maintained by the flash driver 306. The data
25 structures are stored in a volatile portion of memory 304. The one or more

1 secondary tables 1004, 1006 contain mappings of logical-to-physical sector
2 addresses. Each of the secondary data structures 1004, 1006, as will be explained,
3 have a predetermined capacity of mappings. The master data structure 1002
4 contains a pointer to each of the one or more secondary data structures 1004, 1006.
5 Each secondary data structure is allocated on as needed basis for mapping those
6 logical-to-physical addresses that are used to store data. Once the capacity of a
7 secondary data structure 1004, 1006, etc., is exceeded, another secondary data
8 structure is allocated, and another, etc., until eventually all possible physical sector
9 addresses on the flash medium 100/200 are mapped to logical sector addresses.
10 Each time a secondary table is allocated, a pointer contained in the master data
11 structure 1002 is enabled by the flash driver 306 to point to it.

12 Accordingly, the flash driver 306 dynamically allocates one or more
13 secondary data structures 1004, 1006 based on the amount of permanent data
14 stored on the flash medium itself. The size characteristics of the secondary data
15 structures are computed at run-time using the specific attributes of the flash
16 memory medium 100/200. Secondary data structures are not allocated unless the
17 secondary data structure previously allocated is full or insufficient to handle the
18 amount of logical address space required by the file system 305. Dynamic look-up
19 data structure 1000, therefore, minimizes usage of memory 304. Dynamic look-up
20 data structure 1000 lends itself to computer devices 300 that use calendars,
21 inboxes, documents, etc. where most of the logical sector address space will not
22 need to be mapped to a physical sector address. In these applications, only a finite
23 range of logical sectors are repeatedly accessed and new logical sectors are only
24 written when the application requires more storage area.
25

1 The master data structure 1002 contains an array of pointers, 0 through N
2 that point to those secondary data structures that are allocated. In the example of
3 FIG. 10, the pointers at location 0 and 1 point to secondary data structures 1004
4 and 1006, respectively. Also, in the example illustration of FIG. 10, pointers 2
5 through N do not point to any secondary data structures and would contain a
6 default setting, "NULL", such that the logical-to-physical sector mapping module
7 404 knows that there are no further secondary data structures allocated.

8 Each secondary data structure 1004, 1006 is similar to data structures 600,
9 but only a portion of the total possible medium is mapped in the secondary data
10 structures. The secondary data structures permit the flash abstraction logic 308 to
11 reduce the amount space needed in memory 304, to only those portions of logical
12 sectors addresses issued by the file system. Each secondary data structure is $(b*k)$
13 bytes in size, where k is the number of physical sector addresses contained in the
14 data structure and b is the number of bytes used to store each physical sector
15 address.

16 FIG. 11 illustrates a process 1100 for dynamically allocating look-up data
17 structures for tracking data on the flash memory medium 100/200. Process 1100
18 includes steps 1102 through 1106. The order in which the process is described is
19 not intended to be construed as a limitation. Furthermore, the process can be
20 implemented in any suitable hardware, software, firmware, or combination
21 thereof.

22 In step 1102, a master data structure 1002 containing the pointers to one or
23 more secondary data structures 1004, 1006 is generated. The master data structure
24 1002 in this exemplary implementation is fixed in size. At the time the computer
25 device 300 boots-up, the flash medium logic 310 determines the size of the flash

memory medium 100/200 and relays this information to the flash abstraction logic 308. Based on the size of the flash medium, the flash abstraction logic 308 calculates a range of physical addresses. That is, suppose the size of the flash medium is 16 MB, then a NAND flash medium 100 will typically contain 32768 sectors each 512 bytes in size. This means that the flash abstraction logic 308 may need to map a total of 0 through 32768 logical sectors in a worse case scenario, assuming all the memory space is used on the flash medium. Knowing that there are 2^{15} sectors on the medium, the flash abstraction logic 308 can use 2 bytes to store the physical sector address for each logical sector address. So the master data structure is implemented as an array of 256 DWORDs ($N=256$), which covers the maximum quantity of logical sector addresses (e.g., 32768) to be issued by the files system. So, there are a total of 256 potential secondary data structures.

In step 1104 the secondary data structure(s) are allocated. First, the flash abstraction logic determines the smallest possible size for each potential secondary data structure. Using simple division, $32768 / 256 = 128$ logical sector addresses supported by each data structure. As mentioned above, the entire physical space can be mapped using 2 bytes, $b=2$, therefore, each secondary data structure will be 256 bytes in size or ($b=2 * k=128$).

Now, knowing the size of each secondary data structure, suppose that the file system 305 requests to write to logical sector addresses 50-79, also known as LS50-LS79. To satisfy the write requests from the files system 305, the flash abstraction logic 308 calculates that the first pointer in master data structure 1002 is used for logical sector addresses LS0-LS127 or data structure 1004. Assuming the first pointer is NULL, the flash abstraction logic 308 allocates data structure 1004 (which is 256 bytes in size) in memory 304. As indicated in step 1106, the

1 flash abstraction logic 308 enables the pointer in position 0 of the master data
2 structure to point to data structure 1004. So, in this example, data structure 1004
3 is used to store the mapping information for logical sectors LS50-LS79.

4 The flash abstraction logic 308 allocates a secondary data structure, if the
5 file system 305 writes to the corresponding area in the flash medium 100/200.
6 Typically, only the logical sector addresses that are used are mapped by the flash
7 abstraction logic 308. So, in the worst case scenario, when the file system 305
8 accesses the entire logical address space, then all 256 secondary data structures
9 (only two, 1004, 1006 are shown to be allocated in the example of FIG. 10), each
10 256 bytes in size will be allocated requiring a total of 64KB of space in memory
11 304.

12 When an allocated data structure 1004, for instance, becomes insufficient to
13 store the logical sector address space issued by the file system 305, then the flash
14 abstraction logic 308 allocates another data structure, like data structure 1006.
15 This process of dynamically allocating secondary data structures also applies if
16 data structure 1004 becomes sufficient at a later time to again handle all the logical
17 sector address requests made by the file system. In this example, the pointer to
18 data structure 1006 would be disabled by the flash abstraction logic 308; and data
19 structure 1006 would become free space in memory 304.

20 ***Uniform Wear Leveling and Recycling of Sectors***

21 FIG. 12 is a diagram of flash memory medium 100/200 viewed and/or
22 treated as a continuous circle 1200 by the flash driver 306. Physically the flash
23 memory media is the same as either media 100/200 shown in FIGS. 1 and 2,
24 except the flash abstraction logic 308, organizes the flash memory medium as if it
25 is a continuous circle 1200, containing 0-to-N blocks. Accordingly, the highest

1 physical sector address (individual sectors are not shown in FIG. 12 to simplify the
2 illustration, but may be seen in FIGS. 1 and 2) within block N and the lowest
3 physical sector address within block 0 are viewed as being contiguous.

4 FIG. 13 illustrates another view of media 100/200 viewed as a continuous
5 circle 1200. In this exemplary illustration, the sector manager 402 maintains a
6 write pointer 1302, which indicates a next available free sector to receive data on
7 the medium. The next available free sector is a sector that can accept data without
8 the need to be erased first in a prescribed order. The write pointer 1102 is
9 implemented as a combination of two counters: a sector counter 1306 that counts
10 sectors and a block counter 1304 that counts blocks. Both counters combined
11 indicate the next available free sector to receive data.

12 In an alternative implementation, the write pointer 1302 can be
13 implemented as a single counter and indicate the next physical sector that is free to
14 accept data during a write operation. According to this implementation, the sector
15 manager 402 maintains a list of all physical sector addresses free to receive data
16 on the medium. The sector manager 402 stores the first and last physical sector
17 addresses (the contiguous addresses) on the medium and subtracts the two
18 addresses to determine an entire list of free sectors. The write pointer 1302 then
19 advances through the list in a circular and continuous fashion. This reduces the
20 amount of information needed to be stored by the sector manager 402.

21 FIG. 14 illustrates a process 1400 used by the sector manager 402 to
22 determine the next available free sector location for the flash driver 306 to store
23 data on the medium 100/200. Process 1400 also enables the sector manager 402 to
24 provide each physical sector address (for the next free sector) for assignment to
25 each logical sector address write request by the file system 305 as described

1 above. Process 1400 includes steps 1402-1418. The order in which the process is
2 described is not intended to be construed as a limitation. Furthermore, the process
3 can be implemented in any suitable hardware, software, firmware, or combination
4 thereof.

5 In step 1402, the X block counter 1304 and Y sector counter 1306 are
6 initially set to zero. At this point it is assumed that no data resides on the medium
7 100/200.

8 In step 1404, the driver 306 receives a write request and the sector manager
9 402 is queried to send the next available free physical sector address to the logical-
10 to-physical sector mapping module 404. The write request may come from the file
11 system 305 and/or internally from the compactor 406 for recycling sectors as shall
12 be explained in more detail below.

13 In step 1406, the data is written to the sector indicated by the write pointer
14 1302. Since both counters are initially set to zero in this exemplary illustration,
15 suppose that the write pointer 1302 points to sector zero, block zero.

16 In step 1408, the sector counter 1306 is advanced one valid sector. For
17 example, the write pointer advances to sector one of block zero, following the
18 example from step 1406.

19 Next, in decisional step 1410, the sector manager 402 checks whether the
20 sector counter 1306 exceeds the number of sectors K in a block. If the Y count
21 does not exceed the maximum sector size of the block, then according to the NO
22 branch of decisional step 1410, steps 1404-1410 repeat for the next write request.

23 On the other hand, if the Y count does exceed the maximum sector size of
24 the block, then the highest physical sector address of the block was written to and
25 the block is full. Then according to the YES branch of step 1410, in step 1412 the

1 Y counter is reset to zero. Next, in step 1414, X block counter 1304 is
2 incremented by one, which advances the write pointer 1302 to the next block at
3 the lowest valid physical sector address, zero, of that block.

4 Next, in decisional step 1416, the compactor 406 checks whether the X
5 block counter is pointing to a bad block. If it is, X block counter 1304 is
6 incremented by one. In one implementation, the compactor 406 is responsible for
7 checking this condition. As mentioned above, the sector manager stores all of the
8 physical sector addresses that are free to handle a write request. Entire blocks of
9 physical sector addresses are always added by the compactor during a compaction
10 or during initialization. So, the sector manager 402 does not have to check to see
11 if blocks are bad, although the sector manager could be implemented to do so. It
12 should also be noted that in other implementations step 1416 could be performed
13 at the start of process 1400.

14 In step 1417, the X block counter 1304 is incremented until it is pointing to
15 a good block. To avoid a continuous loop, if all the blocks are bad, then process
16 1400 stops at step 1416 and provides an indication to a user that all blocks are bad.

17 Next in decisional step 1418, the sector manager checks whether the X
18 block counter 1304 exceeds the maximum numbers of blocks N. This would
19 indicate that write pointer 1302 has arrived full circle (at the top of circle 1200). If
20 that is the case, then according to the YES branch of step 1418, the process 1400
21 repeats and the X and Y counter are reset to zero. Otherwise, according to the NO
22 branch of step 1418, the process 1400 returns to step 1404 and proceeds.

23 In this exemplary process 1400, the write pointer 1302 initially starts with
24 the lowest physical sector address of the lowest addressed block. The write
25 pointer 1302 advances a sector at a time through to the highest physical sector

1 address of the highest addressed block and then back to the lowest, and so forth.
2 This continuous and circular process 1400 ensures that data is written to each
3 sector of the medium 100/200 fairly and evenly. No particular block or sector is
4 written to more than any other, ensuring even wear-levels throughout the medium
5 100/200. Accordingly, process 1400 permits data to be written to the next
6 available free sector extremely quickly without expensive processing algorithms
7 used to determine where to write new data while maintaining even wear-levels.
8 Such conventional algorithms can slow the write speed of a computer device.

9 In an alternative implementation, it is possible for the write pointer 1302 to
10 move in a counter clock wise direction starting with highest physical sector
11 address of the highest block address N and decrement its counters. In either case,
12 bad blocks can be entirely skipped and ignored by the sector manager.
13 Additionally, the counters can be set to any value and do not necessarily have to
14 start with the highest or lowest values of for the counters.

15 FIG. 15 illustrates another view of media 100/200 viewed as a continuous
16 circle 1200. As shown in FIG. 15, the write pointer 1302 has advanced through
17 blocks 0 through 7 and is approximately half way through circle 1200.
18 Accordingly, blocks 0 through 7 contain dirty, valid data, or bad blocks. That is,
19 each good sector in blocks 0 through 7 is not free, and therefore, not available to
20 receive new or modified data. Arrow 1504 represents that blocks 0 through 7
21 contain used sectors. Eventually, the write pointer 1302 will either run out of free
22 sectors to write to unless sectors that are marked dirty or are not valid are cleared
23 and recycled. To clear a sector means that sectors are reset to a writable state or in
24 other words are "erased." In order to free sectors it is necessary to erase at least a
25 block at a time. Before a block can be erased, however, the contents of all good

1 sectors are copied to the free sectors to a different portion of the media. The
2 sectors are then later marked "dirty" and the block is erased.

3 The compactor 406 is responsible for monitoring the condition of the
4 medium 100/200 to determine when it is appropriate to erase blocks in order to
5 recycle free sectors back to the sector manager 402. The compactor 406 is also
6 responsible for carrying out the clear operation. To complete the clear operation,
7 the compactor 406, like the sector manager 402, maintains a pointer. In this case,
8 the compactor 406 maintains a clear pointer 1502, which is shown in FIG. 15. The
9 clear pointer 1502 points to physical blocks and as will be explained enables the
10 compactor 406 to keep track of sectors as the medium 100/200 as blocks are
11 cleared. The compactor 406 can maintain a pointer to a block to compact next
12 since an erase operation affects entire blocks. That is, when the compactor 406 is
13 not compacting a block, the compactor 406 points to a block.

14 FIG. 16 is a flow chart illustrating a process 1600 used by the compactor to
15 recycle sectors. Process 1600 includes steps 1602-1612. The order in which the
16 process is described is not intended to be construed as a limitation. Furthermore,
17 the process can be implemented in any suitable hardware, software, firmware, or
18 combination thereof. In step 1602, the compactor 406 monitors how frequently
19 the flash memory medium 100/200 is written to or updated by the file system.
20 This is accomplished by specifically monitoring the quantities of free and dirty
21 sectors on the medium 100/200. The number of free sectors and dirty sectors can
22 be determined counting free and dirty sectors stored in tables 600 and/or 900
23 described above.

24 In decisional step 1604, the compactor 406 performs two comparisons to
25 determine whether it is prudent to recycle sectors. The first comparison involves

1 comparing the amount of free sectors to dirty sectors. If the amount of dirty
2 sectors outnumber the free sectors, then the compactor 406 deems it warranted to
3 perform a recycling operation, which in this case is referred to as a "service
4 compaction." Thus a service compaction is indicated when the number of dirty
5 sectors outnumber the quantity of free sectors.

6 If a service compaction is deemed warranted, then in step 1606 the
7 compactor waits for a low priority thread 1606, before seizing control of the
8 medium to carry out steps 1608-1612 to clear blocks of dirty data. The service
9 compaction could also be implemented to occur at other convenient times when it
10 is optional to recycle dirty sectors into free sectors. For instance, in an alternative
11 implementation, when one third of the total sectors are dirty, the flash abstraction
12 logic 308 can perform a service compaction. In either implementation, usually the
13 compactor 406 waits for higher priority threads to relinquish control of the
14 processor 302 and/or flash medium 100/200. Once a low priority thread is
15 available, the process proceeds to step 1608.

16 Referring back to step 1604, the second comparison involves comparing the
17 amount of free sectors left on the medium, to determine if the write pointer 1302 is
18 about to or has run out of free sectors to point to. If this is the situation, then the
19 compactor 406 deems it warranted to order a "critical compaction" to recycle
20 sectors. The compactor does not wait for a low priority thread and launches
21 immediately into step 1608.

22 In step 1608, the compactor 406 operates at either a high priority thread or
23 low priority thread depending on step 1604. If operating at a high level thread
24 (critical compaction), the compactor 1102 is limited to recycling a small number,
25 e.g., 16 dirty sectors, into free sectors and return control of the processor back to

1 computer device 300 to avoid monopolizing the processor 302 during such an
2 interruption.

3 Thirty two sectors per block is commonly manufactured for flash media,
4 but other numbers of sectors, larger or smaller, could be selected for a critical
5 compaction. Regardless of these size characteristics, the number of sectors
6 recycled during a critical compaction is arbitrary but must be at least 1 (in order to
7 satisfy the current WRITE request). A critical compaction stalls the file system
8 305 from being able to complete a write; therefore, it is important to complete the
9 compaction as soon as possible. In the case of a critical compaction, the
10 compactor 406 must recycle at least one dirty sector into a free sector so that there
11 is space on the medium to fulfill the pending write request. Having more than one
12 sector recycled at a time, such as 16, avoids the situation where there are multiple
13 pending write requests and multiple critical compactations that are performed back-
14 to-back, effectively blocking control of the processor indefinitely. So, while the
15 number of sectors recycled chosen for a critical compaction can vary, a number
16 sufficient to prevent back-to-back critical compactations is implemented in the
17 exemplary description.

18 So, in step 1608, the compactor 406 will use the clear pointer 1502 to scan
19 sectors for valid data, rewrite the data to free sectors, and mark a sector dirty after
20 successfully moving data. Accordingly, when moving data, the compactor uses
21 the same processes described with reference to process 700, which is the same
22 code that is used when the file system 305 writes new and/or updates data. The
23 compactor 406 queries the sector manager 402 for free sectors when moving data,
24 in the same fashion as described with reference to process 1400.

1 In step 1610, the compactor 406 moves the clear pointer 1502 sector-by-
2 sector using a sector counter like the write counter 1306 shown in Fig. 13, except
3 this sector counter pertains to the location of the clear pointer 1502. The
4 compactor 406 also keeps track of blocks through a counter in similar fashion as
5 described with reference to the write pointer 1302. However, the amount of
6 blocks cleared is determined by the number of dirty sectors with the exception of a
7 critical compaction. In a critical compaction, the compactor only compacts
8 enough blocks to recycle a small number of physical sectors (i.e. 16 sectors).

9 In step 1612, the compactor erases (clears) those blocks which contain good
10 sectors that are fully marked dirty. FIG. 17 shows exemplary results from process
11 1600. In this example, blocks 0 and 1 were cleared and the clear pointer was
12 moved to the first sector of block 2, in the event another compaction is deemed
13 warranted. As a result, the compactor 406 recycled two blocks worth of the
14 sectors from blocks 0 and 1, which provides more free sectors to the sector
15 manager 402. Used sectors 1504 forms a data stream (hereinafter a "data stream"
16 1504) that rotates in this implementation in a clockwise fashion. The write pointer
17 1302 remains at the head of the data stream 1504 and the clear pointer 1502
18 remains at the end or "tail" of the data stream 1504. The data stream 1504 may
19 shrink as data is deleted, or grow as new data is added, but the pointers always
20 point to opposite ends of the data stream 1504: head and tail.

21 Treating the flash memory medium as if the physical sector addresses form
22 a continuous circle 1200, and using the processes described above, enables the
23 flash abstraction logic 308 to accomplish uniform wear-leveling throughout the
24 medium 100/200. The compactor 406 selects a given block the same number
25 times for recycling of sectors through erasure. Since flash blocks have a limited

1 write/erase cycle, the compactor as well as the sector manager distributes these
2 operations across blocks 0-N as evenly and as fairly as possible. In this regard, the
3 data stream 1504 rotates in the circle 1200 (i.e. the medium 100/200) evenly
4 providing perfect wear-levels on the flash memory medium 100/200.

5 In the event of power failure, the flash abstraction logic 310 contains
6 simple coded logic that scans the flash memory medium 100/ 200 and determines
7 what locations are marked free and dirty. The logic is then able to deduce that the
8 data stream 1504 resides between the locations marked free and dirty, e.g., the
9 data stream 1106 portion of the circle 1200 described in FIG. 17. The head and
10 tail of the data stream 1504 is easily determined by locating the highest of the
11 physical sector addresses containing data for the head and by locating the lowest
12 of the physical sector addresses containing data for the tail.

13 *NOR Flash Devices*

14 Although all the aforementioned sections in this Detailed Description
15 section apply to NAND and NOR flash devices, if a NOR flash memory medium
16 200 is used, some additional implementation is needed for the flash medium logic
17 to support the storing of data in each physical sector on the medium 200. Each
18 NOR block 0, 1, 2, etc. can be treated like a NAND flash memory medium 100, by
19 the flash medium logic 310. Specifically, each NOR block is subdivided into
20 some number of pages where each page consists of a 512 byte "data area" for
21 sector data and an 8 byte "spare area" for storing things like to the logical sector
22 address, status bits, etc. (as described above).

23 FIG. 18 illustrates a logical representation of a NOR flash memory medium
24 200 divided in way to better support the processes and techniques implemented by
25 the flash driver. In this implementation, sectors 1802 contain a 512 byte data area

1 1803 for the storage of sector related data and 8 bytes for a spare area 1804.
2 Sections 1806 represent unused portions of NOR blocks, because a NOR Flash
3 block is usually a power of 2 in size, which is not evenly divisible. For instance,
4 consider a 16 MB NOR flash memory device that has 128 flash blocks each 128
5 KB in size. Using a page size equal to 520 bytes, each NOR flash block can be
6 divided into 252 distinct sectors with 32 bytes remaining unused. Unfortunately,
7 these 32 bytes per block are "wasted" by the flash medium logic 310 in the
8 exemplary implementation and are not used to store sector data. The tradeoff,
9 however, is the enhanced write throughput, uniform wear leveling, data loss
10 minimization, etc. all provided by the flash abstraction logic 308 of the exemplary
11 flash driver 306 as described above. Alternative implementations could be
12 accomplished by dividing the medium 200 into different sector sizes.

13 *Computer readable Media*

14
15 An implementation of exemplary subject matter using a flash driver as
16 described above may be stored on or transmitted across some form of computer-
17 readable media. Computer-readable media can be any available media that can be
18 accessed by a computer. By way of example, and not limitation, computer
19 readable media may comprise "computer storage media" and "communications
20 media."

21 "Computer storage media" include volatile and non-volatile, removable and
22 non-removable media implemented in any method or technology for storage of
23 information such as computer readable instructions, data structures, program
24 modules, or other data. Computer storage media includes, but is not limited to,
25 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,

1 digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic
2 tape, magnetic disk storage or other magnetic storage devices, or any other
3 medium which can be used to store the desired information and which can be
4 accessed by a computer.

5 "Communication media" typically embodies computer readable
6 instructions, data structures, program modules, or other data in a modulated data
7 signal, such as carrier wave or other transport mechanism. Communication media
8 also includes any information delivery media.

9 The term "modulated data signal" means a signal that has one or more of its
10 characteristics set or changed in such a manner as to encode information in the
11 signal. By way of example, and not limitation, communication media includes
12 wired media such as a wired network or direct-wired connection, and wireless
13 media such as acoustic, RF, infrared, and other wireless media. Combinations of
14 any of the above are also included within the scope of computer readable media.

15 **Conclusion**

16 Although the invention has been described in language specific to structural
17 features and/or methodological acts, it is to be understood that the invention
18 defined in the appended claims is not necessarily limited to the specific features or
19 acts described. Rather, the specific features and acts are disclosed as exemplary
20 forms of implementing the claimed invention.